1988

Multilevel Asynchronous Iteration for PDEs

Dan C. Marinescu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
88-820

MULTILEVEL ASYNCHRONOUS ITERATION FOR PDES

Dan C. Marinescu
J. R. Rice

CSD-TR 820
November 1988

MULTI–LEVEL ASYNCHRONOUS ITERATION FOR PDES

*Dan C. Marinescu**
*John R. Rice**
Department of Computer Science
Purdue University
Technical Report CSD-TR-820
CAPO Report CER-88-35

December 1, 1988

1 INTRODUCTION

Modeling and analysis of synchronization in parallel computing raises difficult questions. Empirical data are largely unavailable, due to the present state of the art in the instrumentation of parallel systems. Only measurements related to the aggregate program behavior such as total execution time, processor utilization, etc., can be carried out routinely, while detailed data concerning communication and synchronization costs are usually unavailable. There are cases in parallel processing when synchronization cannot be avoided, for example in case of iteration techniques for partial differential equations (PDEs) [3], [5].

We have developed a non-deterministic model for parallel computation [1] which shows that in the general case the overhead associated with synchronization depends principally upon two factors, namely, the number of PEs running in parallel, and the actual distribution of the execution time on PEs. For particular distributions the overhead associated with synchronization is independent of the number of PEs running in parallel when this number is large, hence massive parallelism does not become prohibitively expensive solely due to synchronization. This is the case for the uniform distributions, when only the coefficient of variation of the distribution determines the synchronization overhead. For other distributions, like the exponential one, the synchronization overhead grows logarithmically in the number of PEs.

In a recent paper [4] we have analyzed some aspects of the behavior of a two level asynchronous algorithm for solving PDEs. The model develped in [1] has been specialized to this case and we explored several issues concerning the behavior of this algorithm on a

*Work supported in part by the Strategic Defense Initiative under Army Research Office contract DAAL03-86-K-0106.

hypercube machine. These issues include how to partition the hypercube, how to allocate processors to the two levels, the effects of synchronization, and how to relate the grids on the two levels. Our algorithm is designed to use less communication than is "natural" for such a computation because communication is so expensive on the present generation of hypercube machines. The general results of [4] is that the approach looks promising.

In this paper we extend the algorithm and its analysis to multiple levels (Section 2), study the effects of various iterations chosen within levels (Sections 4 and 5), and provide a technique for efficiently embedding the multiple levels in the hypercube (Section 6).

2 MULTIPLE LEVEL ASYNCHRONOUS PDE ALGORITHMS

Consider the PDE problem $Lu = f$ on the domain $D = [0,1]\times[0,1]$ with Dirichlet boundary conditions. We first subdivide D to obtain *level 1* with $N(1)$ overlapping domains

$$D_k = [0,1] \times [(k-1)/N(1), \quad k/N(1)], k = 1,2,...,N(1).$$

We then subdivide each D_k into $N(2)/N(1)$ domains

$$D_{kj} = [0,1] \times [(k-1)/N(1) + (j-1)\delta, (k-1)/N(1) + j\delta],$$

$$j = 1,2,...,N(2)/N(1) \quad .$$

where $\delta = 1/N(2)$. This determines *level 2* with $N(2)$ domains. We now formalize this with

Definition 1: (*Multi-level Structure*) A multi-level structure is an interconnection of M linear arrays $L_1,...,L_M$ each representing a level of the structure. Each linear array is subdivided into subdomains. D_{ij} is the jth subdomain in level i. The number $N(p)$ of subdomains at level p is given by

$$N(p) = [R(1) \cdot R(2)...R(j)...R(p-1)]N(1) \text{ for } 1 \leq p \leq M-1, \text{ and } 1 \leq j \leq p-1$$

In the previous expression

- $N(i)$ is the number of subdomains at level i,
- $R(j)$ is the domain refinement factor at level j defined as, $R(j) = \frac{N(j+1)}{N(j)}$

Each subdomain D_{ij} at level i is connected to its neighbors $D_{i,j-1}$ and $D_{i,j+1}$, if $1 < j < N(i)$. $D_{i,1}$ and $D_{i,N(i)}$ are connected with only one neighbor $D_{i,1}$ and $D_{i,N(i)-1}$ respectively.

In addition a subdomain $D_{i,j}$ at level $i, 1 \leq j \leq N(i)$, is connected to f_i subdomains at level $i+1$, for $1 \leq i \leq M-2$. The number f_i is called the *fan-out index*. An example of a multi-level structure is given in [3] in connection with a two level asynchronous algorithm for solving PDEs. Figure 1 shows this structure for $M = 2$, $N(0) = 4$, and $q_0 = 3$. The fan-out index is in this case $f_1 = 2$, a domain $D_{1,k}$ at level 1 has to communicate with two domains $D_{2,j}$ and $D_{2,j+1}$ at level 2.

We discretize the linear PDE on each level, using $nx(1)$ x-points on level 1 and $nx(p+1) = q(p)nx(p)$ x-points on level $p+1$. The y discretization is the same. An iteration method is then used to solve the resulting linear system on each level. We anticipate a parallel implementation of the iteration with $N(p)$ processors assigned to level p. We see that there are two nested structures here. First is that of the domains, each of which is to be assigned to a single processor. Second is that of the grids which determine the sizes of the linear systems for each domain (processor) and the error in discretizing the PDE. These structures may be visualized by the following parameters:

Level	Number of Domains	Domain Refinement Factor	Number of x-grid Points	Grid Size Ratio
1	N(1)	R(1)	nx(1)	q(1)
–	–	–	–	–
p	N(p)	R(p)	nx(p)	q(p)

We have

$$N(p+1) = R(p)N(p), \qquad nx(p+1) = q(p)nx(p)$$

For simplicity, we ignore for now the fact that the domains and grid sizes might not mesh nicely on the various levels.

In [4] we present a rationale for this multi-level iteration in the case of two levels. We note here merely its principal properties, namely

1. The iterations on different levels operate asynchronously,
2. The iterations within levels are synchronized among the domains (processors) on that level,
3. Information may be passed between levels along the domain boundaries,
4. These are formulas to transfer information between different grid sizes (interpolate when going to a finer grid, smooth when going to a coarser grid).

The structure of this computation includes multi-level algorithms such as multigrid or nested iteration. It is practical even to have different "types" of iterations on different levels. At this point we do not differentiate between all these possibilities.

We have analyzed this computation with models that assume rates of convergence for the linear system which range from those of Gauss-Seidel (slow) to SOR with optimum parameter (fast), to multigrid (very fast). We have also analyzed the effect of different discretizations, namely, second order and fourth order. See Section 4 for more details of the performance model and analysis.

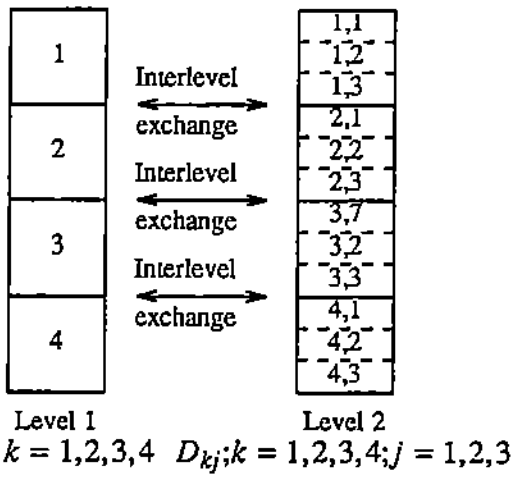

Figure 1. A two level partition of D with $N(1)$ and $N(2)$ domains respectively. The algorithm has information passed asynchronously to and from level 2 along the boundaries of the domains of level 1.

3 A UNIFIED MODEL OF PARALLEL COMPUTATION

In [4] we have specialized the model of [1] to the case of a two level asynchronous iteration. We recall here the basic form of this model and extend it to multiple levels. We view a parallel computation as a sequence of w synchronization epochs each using I_i processors, and we express the expected execution time of a parallel computation as

$$T^* = E(T_c) = \alpha + w\beta + \sum_{i=1}^{w} \mu_i(1 + \Delta_i)$$

In this expression α represents the cost due to control of the parallel computation. The communication costs are denoted by β. Note that β does reflect only the cost of communication which cannot be overlapped with computations. The final term is the execution time of the I_i processors active in any epoch i, whose execution times are independent, identically distributed random variables with mean μ_i, and variance σ_i. The average cost attributed to the load imbalance in epoch i is $\Delta_i = \Delta(C_i, Ii)$, with $C_i = \frac{\sigma_i}{\mu_i}$.

4 MODEL OF MULTI–LEVEL ITERATION ON A HYPERCUBE MACHINE

We now model the multi-level iteration algorithm and its implementation on a hypercube. The model is constructed in two phases. First we study the algorithm and compare it with a standard, one level algorithm. We define the *algorithmic speed up* as the ratio of the execution time of the one level algorithm and the execution time of the multi-level algorithm on the same abstract machine.

Then we model the actual implementation of the algorithm with M levels on a parallel machine with a hypercube architecture, the NCUBE. We define the *actual speed up* as the ratio of the execution time on the NCUBE model when level M runs alone as compared with the execution time when all levels execute concurrently and lower levels feed information to the higher ones as prescribed by our algorithm.

For the algorithmic model, let us use the following notations:

- S – algorithmic speed up,
- m_B – number of iterations required by a one level PDE algorithm (using level M),
- I_B – execution time per iteration for the one level algorithm on the hypothetical machine,
- m_A – number of iterations required by the multiple level PDE algorithm,
- I_A – execution time per iteration for the multiple level algorithm on the same hypothetical machine.

Then, the *algorithmic speed up* is

$$S = \frac{m_B I_B}{m_A I_A}$$

As a first approximation $I_A = I_B$ and S becomes m_B/m_B, the ratio of the number of iterations required.

The second model involves a more detailed account of computation, communication and synchronization at each level. To describe the model, the following notations are used to describe the machines and the computational structure.

- t_a – time for arithmetic operations per iteration at a single grid point, a typical value for t_a is 5,
- f_{sy} – synchronization factor (effect of synchronizing within levels),
- t_c – communication cost per unit of data, and typical value is $t_c = 2$,
- t_s – start-up time for a communication act, a typical value for t_s is 200,
- N – total number of processors,
- $N(p)$ – number of processors assigned to level p,
- $R(p)$ – domain refinement factor,
- M – total number of levels

The model of the numerical aspects of the computation includes the error behavior parameters:

$$ERR(p) = \text{Discretization error on level } p = [nx(0) + nx(p)]^{-order}$$

where

order	=	order of the discretization error (usually 2, 3 or 4)
$nx(0)$	=	number of x-points before any convergence toward the PDE solution starts
$nx(p)$	=	number of x-grid points at level p,
$q(p)$	=	grid refinement factor $nx(p+1) = q(p)nx(p)$,

We assume an initial guess for the PDE which has an error equal to 1.0. This model also includes the iteration behavior parameters

m_B	=	iteration count required by the one level iteration
$m(p,j)$	=	count required by the iteration on level p in Phase j (= 1 or 2) of the iteration
$r(p,j)$	=	convergence rate of the iteration on level p in Phase j of the iteration

The iteration count m, convergence ratio r and discretization error ERR, are related by

$$r^m = ERR$$

The phases of the iteration on level p are defined as follows:

Phase 1: The error in the linear systems on level p exceeds the discretization error on level $p-1$. The group of subdomains from a single level $p-1$ domain are treated independently using boundary values transferred down from level $p-1$. Thus $R(p-1)$ subdomains are a unit.

Phase 2: The error in the linear systems on level p is less than the discretization error on level $p-1$. All the subdomains on level p are then treated as a unit.

The convergence rates are assumed to depend on the number of x-points in the subdomains treated as a unit and we model this behavior by

$$r = 1 - 1/(nx)^k$$

where nx is the number of x-points and k is a parameter of the iteration ($k = 2$ corresponds to Gauss-Seidel iteration, $k = 1$ to SOR and "faster" iterations may be approximately modeled by fractional values). The convergence rate formulas that result are

$$\begin{aligned} r(p,1) &= 1 - 1/(nx(p)/N(p))^k \\ r(p,2) &= 1 - 1/nx(p)^k \end{aligned}$$

The number $m(p,1)$ of iterations for Phase 1 is clearly more than (a) the total $m(p-1,1)+m(p-1,2)$ for level $p-1$ adjusted by the factor $R(p-1)/q(p-1)$ relating (approximately) the time per iteration on each level. It is also plausible that $m(p,1)$ is more than (b) the number required to reduce the error from the initial guess 1.0 to the discretization error level $ERR(p-1)$. It is also clear that $m(p,1)$ is less than the sum of these two numbers. We assume that $m(p,1)$ is equal to the maximum of the two. Normally the latter is the larger but this is not always the case. Thus we have

$$m(p,1) \;=\; max[i_a, i_b]$$

where

$$\begin{aligned} i_a &= [m(p-1,1)+m(p-1,2)]R(p-1)/q(p-1) \\ r(p,1)^{i_b} &= [nx(0)+nx(p-1)]^{-order} \end{aligned}$$

We can now relate the iteration counts to the grids as follows:

$$r(p,2)^{m(p,2)} = [(nx(0)+nx(p))/(nx(0)+nx(p-1))]^{-order}$$

and obtain

$$i_b \;=\; \frac{-(order)\,\log\,[nx(0)\;+\;nx(p-1)]}{k\,\log\,[1\;-\;N(p)/nx(p)]}$$

$$m(p,1) \;=\; \max\,[i_a, i_b]$$

$$m(p,2) \;=\; \frac{-\,(order)\,\log\,[(nx(0)\;+\;nx(p))/(nx(0)\;+\;nx(p-1)]}{k\,\log\,[1\;-\;1/nx(p)]}$$

The same analysis leads to a simpler formula for m_B, namely

$$m_B \;=\; -\,(order)\,\log\,[nx(0)\;+\;nx(M)]/(k\,\log\,[1\;-\;1/nx(M)])$$

The time T_B for the execution of the one level algorithm is the sum of T_B^a (the arithmetic time) and T_B^c (the communication time). Thus we have

$$\begin{aligned} T_B &= T_B^a + T_B^c \\ &= m_B\left[\,\tfrac{nx(M)^2}{N}\cdot t_a\cdot f_{sy}\,\right] + m_B\left[\,(nx(M)+\log_2 N)\cdot 2t_c \;+\; (1+\log_2 N)\cdot t_s\,\right] \end{aligned}$$

The time T_A for the execution of the multiple level algorithm is also the sum of T_A^a (the arithmetic time) and T_A^c (the communication time). Thus we have

$$\begin{aligned} T_A \;=\; T_A^a + T_A^c \;=\; [m(M,1) \;+\; m(M,2)]\left[\,\tfrac{nx(M)^2}{N(M)}\cdot t_a\cdot f_{sy}\,\right] \\ +\;\; [m(M,1) \;+\; m(M,2)]\;[\,(nx(M) \;+\; \log_2 N(M))\cdot 2t_c \;+\; (1+\log_2 N(M))\cdot t_s\,] \end{aligned}$$

The valuc for $m(M,1)$, of course, depends on all the previous $m(p,j)$ values and these must be computed sequentially starting with $m(1,1) \;=\; 0$ and $m(1,2)$.

The *actual speedup* of the algorithm is defined as T_B/T_A.

5 MAPPING MULTI-LEVEL STRUCTURES ONTO A HYPERCUBE

There are two broad classes of MIMD architectures, the shared memory, and the ensemble architecture. The latter consist of a large number of identical processors interconnected either by a network with fixed topology, or by a switch. In this type of architecture all communication is done by message passing, there is no shared memory or global synchronization. In the following, we consider only ensemble architectures and interconnection networks with a fixed topology. The network topology is characterized by an interconnection graph $G' = (V', E')$ whose nodes represent the processors of the parallel machine and the edges correspond to the communication links. Examples of such machines are tree machines [8], different hypercubes [6, 7], and so on.

To use efficiently an ensemble architecture with a fixed interconnection topology, it is necessary to map a computation with a given topology to the machine architecture in an optimal way. Since communication and synchronization is done by message passing and communication is quite expensive, an optimal mapping should attempt to minimize the communication costs.

In this section we discuss first the issue of optimal mappings to ensemble architectures with a fixed topology and then we investigate mapping of multi-level structures onto a hypercube.

Definition 2: (*Embedding of a graph*) An embedding of a graph $G = (V, E)$ in a graph $G' = (V', E')$ is a one to one function $f : V \rightarrow V'$.

Definition 3: (*Expansion cost of an embedding*) The *expansion cost* of an embedding f is the ratio of the number of nodes in V' to the number of nodes in V.

Definition 4: (*Cost of an embedding*) The *cost* of an embedding f is the largest distance in G' between images of neighboring nodes in G. With $d(X, Y)$ the distance between X and Y, we have

$$C_f = \max_{(A,B)\epsilon E} d(f(A), f(B))$$

Definition 5: (*Optimal mapping*) Given a computation with the computational graph G, an optimal mapping to a parallel machine with an interconnection graph G' is achieved by an embedding of G into G' with an expansion cost of one and with a cost of one.

Optimal mappings into hypercube architectures have been shown to exist for several computational graphs like: linear structures, multi-dimensional grids [6], as well as for some trees [8]. But it seems reasonable to expert that optimal mappings do not always exist. In the following, we discuss mapping with an expansion cost of one, but with costs larger than one. Such mappings will be called non-optimal mappings. The case of multi-level structures to be discussed later, provides such an example. In such a case, a sub-optimal mapping must be performed. To compare sub-optimal mappings, the concept of an effective cost of a mapping is defined.

Definition 6: (*Effective cost of a non-optimal mapping*) The effective cost of a non-optimal mapping of the computational graph G into a parallel machine with the interconnection graph G', is given by the following function of the embedding f of G into G'

$$C_f^{ef} = \sum_{\text{all } (A,B) \in E} (d(f(A), f(B)) - 1) \cdot i_{AB}$$

with $i_{A,B}$ = the intensity of traffic between the nodes A and B.

Note that the effective cost of non-optimal mapping is the function be to minimized by a sub-optimal mapping strategy.

Definition 7: (*Tearing*) An n-cube along the n-th direction is the process of separating the n-cube into two disjoint $(n-1)$ cubes, one obtained by considering all nodes whose n-th but is zero and the other one with all nodes whose n-th but is one. The two cubes will be denoted as $[n-1]^0$-cube and $[n-1]^1$-cube.

Proposition 1: *There exists a unique renumbering of the nodes of an n-cube such that any path of length 2^{n-1} is transformed into a path connecting all nodes of the $[n-1]^0$-cube.*

Proof: Consider a path of length 2^{n-1} connecting the nodes

$$(A_0, A_1, ..., A_i, ..., A_{2^{n-1}-1})$$

Let us select the renumbering scheme which maps every node on the path given to a node of the $[n-1]^0$-cube as follows. A node with label A_i is mapped to a node with label B_i such that

$$B_i = G_n(i) \text{ for } 0 \leq i \leq 2^{n-1} - 1$$

with $G_n(i)$ the Gray code of order n corresponding to integer i. Note that

$$G_n(i) < 2^{n-1} \text{ if } i < 2^{n-1}$$

This property of Gray nodes results immediately from the recursion defining the Gray codes. It follows that all B_i nodes are located on the $[n-1]^0$-cube.

To complete the renumbering scheme, identify every node $\bar{B}_i = (1, b_{n-2}, ..., b_0)$ the opposite of the node $B_i = (0, b_{n-2}, ..., b_0)$ and label it accordingly.

Definition 8: (*Opposite Paths*) Two paths of length ℓ, $\ell \leq 2^{n-1}$ on an n-cube, $\pi^0 = (A_0, A_1, ..., A_{\ell-1}$ $\pi_1 = (B_0, B_1, ..., B_{\ell-1})$ are called opposite path if and only if:

- (a) All nodes A_i, $0 \leq i \leq \ell - 1$ are located on the $[n-1]^0$-cube and all nodes B_j, $0 \leq j \leq \ell - 1$ are located on the $[n-1]^1$-cube

- (b) The two paths contain pairs of opposite nodes, traversed in reverse order

$$B_j = A_{\ell-j} \quad 0 \leq j \leq \ell - 1.$$

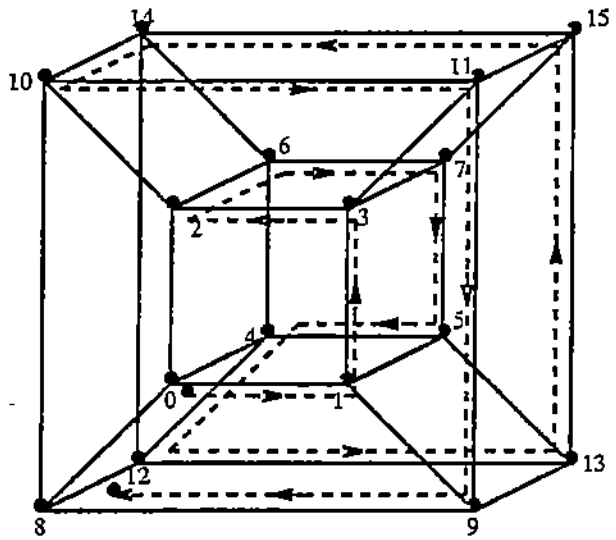

Figure 2. A hypercube of dimension 4 (16 nodes) showing the opposite paths of length 8. One is through nodes 0 to 7 (not in that order) and the other through nodes 8 to 15.

Proposition 2: *A path of length 2^n on an n-cube can be decomposed into two opposite paths of length 2^{n-1}, one covering all nodes of the $[n-1]^0$-cube and the other covering all the nodes of the $[n-1]^1$-cube.*

Proof: A path of length 2^n on an n-cube can be constructed using Gray codes of order n, [3]. Note that there is a one to one correspondence between an integer $i < 2^n$ and its Gray code of order $n, G_n(i)$. The Gray codes of order n are generated recursively as

$$G_n = \{0G_{n-1}, 1G_{n-1}^R\}$$

with G_i^R the reverse of the i-tuple G_i. If $G_i = (g_{i-1}, g_{i-2}, ..., g_0)$ then $G_i^R = (g_0, ..., g_{i-2}, g_{i-r})$.

From the definition of the Gray codes, it follows that

$$G_n(i) < 2^{n-1} \text{ if } i < 2^{n-1}$$

Hence the cycle determined by the Gray codes of order n covers first all the nodes on the $[n-1]^0$-cube, then it covers the nodes of the $[n-1]^1$-cube.

If we consider that the cycle starts at the node labeled 0, the inner path (on the $[n-1]^0$-cube) runs to the node with label 2^{n-2} (since $G_n(2^{n-1}-1) = 2^{n-2}$) then the outer path runs on the $[n-1]^1$-cube starting at the node with label $3 \cdot 2^{n-2}$ (since $G_n(2^{n-1}) = 2^{n-2} + 2^{n-1}$) and terminates at node $2^n - 1$.

Figure 2 illustrates this for $n = 4$. The inner path runs from the node labeled 0 to node 4, and the outer path runs from node 12 to node 8.

The mapping problem is to assign the subdomains of the multi-level structure to the processors of the hypercube. The function f does this by assigning a subdomain D_{ij} to a hypercube node A_k. Note that the inddices i, j and k have the following ranges

$$\begin{aligned} 1 \le i \le M \\ 1 \le j \le N(i) \\ 0 \le k \le 2^m \end{aligned}$$

and we have

$$N(1)\ N(2)\ ...\ N(M) = 2^m$$

In case of multi-level structures, the effective cost of a non-optimal mapping has two components, one due to the intra-level traffic and a second one due to inter-level traffic. If we assume equal traffic intensities, namely i_0 for the inter-level traffic and βi_0 for the intra-level traffic, then the effective cost of a particular mapping has two components

$$C_f^{cf} = C_f^{intra} + C_f^{inter}$$

with

$$C_f^{intra} = \sum_{i=1}^{M-1} \sum_{j=1}^{N(i)-1} (d(f(D_{i,j}), f(D_{i+1,j+1})) - 1)\beta i_0$$

$$C_f^{inter} = \sum_{i=1}^{M-1} \sum_{j=1}^{N(i)-1} \sum_{k=1}^{f_i} (d(f(D_{i,j}), f(D_{i+1,j_k})) - 1) i_0$$

with

- M – number of levels of the structure.
- $N(i)$ – number of subdomains at level i.
- f_i – fan-out index at level i.
- j_k – index of subdomains on level $i+1$ which communicate with subdomain $D_{i,j}$ on level i.

Note that when the fanout $f_i = 1$, then the inter-layer traffic has an effective cost given by

$$C_f^{inter} = \sum_{i=1}^{M-1} \sum_{j=1}^{N(i)-1} (d(f(D_{i,j}), f(D_{i,j+1})) - 1)i_0$$

The "obvious" sub-optimal mapping f_o is obtained by mapping each individual layer as linear structure. In this case,

$$C_{f_0}^{inter} = 0$$

and

$$C_{f_0}^{ef} = C_{f_0}^{inter}$$

Proposition 3: (*Optimal mapping of a multi-level structure*) *Given a multi-level structure with the following properties*

- *it has* 2^n *subdomains,*
- *it has* $M = 2^r$,
- *the fanout indexes are all equal to one,*
- *the number of subdomains at all levels are equal*

$$N(1) = ... = N(M) = 2^{n-r}$$

Then there is an optimal mapping of the multi-level structure into a hypercube of order n.

Proof: Let us denote the levels of the multi-level structure as

$$L_1, ..., L_{2^{r-1}}, L_{2_{r-1}+1}, ..., L_{2^r}$$

Consider now a path of length 2^n determined by the Gray codes of order n. Split this path into 2^r consective paths of length 2^{n-r} identified as

$$C_1, ..., C_{2^{r-1}}, C_{2_{r-1}+1} ..., C_{2^r}$$

Map L_i to C_i for $1 \leq i \leq 2^r$. From Proposition 5 it follows immediately that adjacent levels of the structure are adjacent on the hypercube. Indeed Proposition 5 can be applied recursively and on any hypercube of order 2^{r+1} the two path of length 2^r are opposite hence they consist of 2^r pairs of opposite nodes.

6 ANALYSIS OF THE ITERATION AND ITS PERFORMANCE

The model in Section 4 with the embedding of Section 5 is used to investigate the performance of the multi-level iteration on a hypercube machine (an abstraction of the NCUBE). We study the speed up achieved as the problem size increases and as the parameters of the model vary.

In our earlier study of two level iteration, we determined some optimal values by exhaustively examing all cases, e.g., all ways to divide the processors between two levels. This approach is not practical for more than two levels. We supplement the earlier two level results by more information on the machine parameters and the characteristics of the numerical method.

Figures 3a-3d show the actual speed up obtained for two values of NSTART $= nx(0)^2$, 400 and 2800 using four models for the numerical properties of the iteration

- Gauss-Seidel and second order PDE discretization
- Gauss-Seidel and fourth order PDE discretization
- SOR and second order PDE discretization
- SOR and fourth order PDE discretization

Thus the total number of grid points, $(nx(0) + nx(2))^2$ is varied and we plot the speed up of the two level iteration over the one level iteration. All other parameters of the computation are kept the same (e.g., machine characteristics, convergence rates). All processors assigned to level 1 are idle in the second phase of the two level iteration after transmitting data to level 2. The machine model parameters are $t_s = 200$ and $t_C = 2$ and the synchronization factor uses $C_X = 0.04$. These are typical values for this computation on an NCUBE machine.

There are two curves in Figure 3, one for each of NSTART = 400 and 2800. These overlap in the plots although the speed ups are not identical. These three curves are given:

dashed: for NSTART	=	400
dotted: for NSTART	=	2800
solid: overlap of NSTART	=	400 and 2800

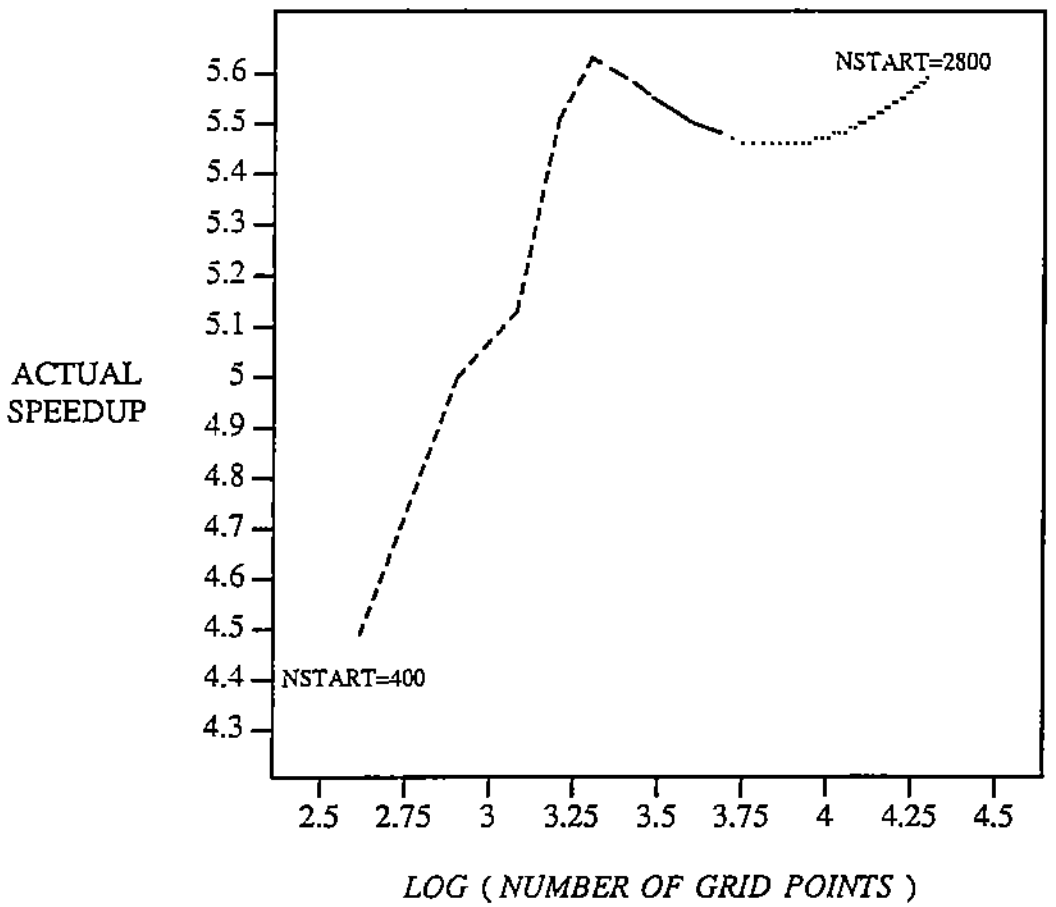

Figure 3a. The *speed up* function of the logarithm of the number of grid points for two values of NSTART, Gauss-Seidel convergence rate, and second order discretization.

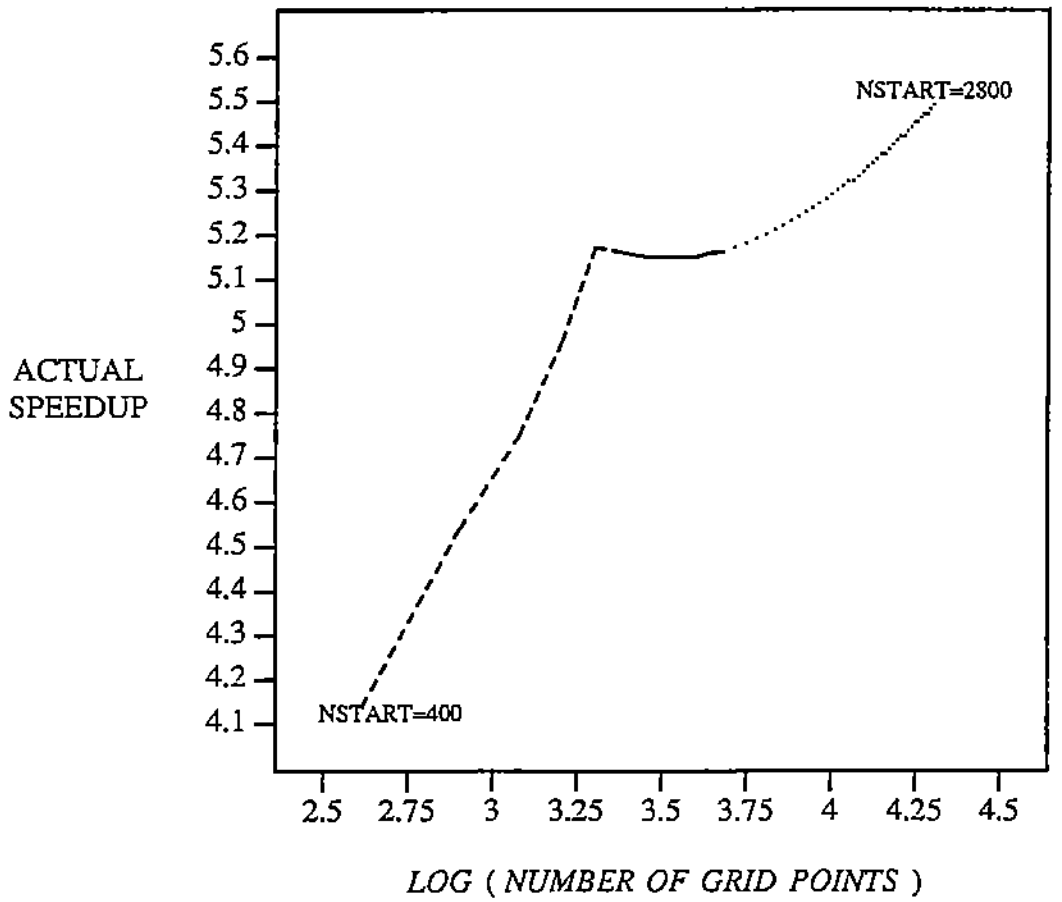

Figure 3b. The *speed up* function of the logarithm of the number of grid points for two values of NSTART, Gauss-Seidel convergence rate, and fourth order PDE discretization error.

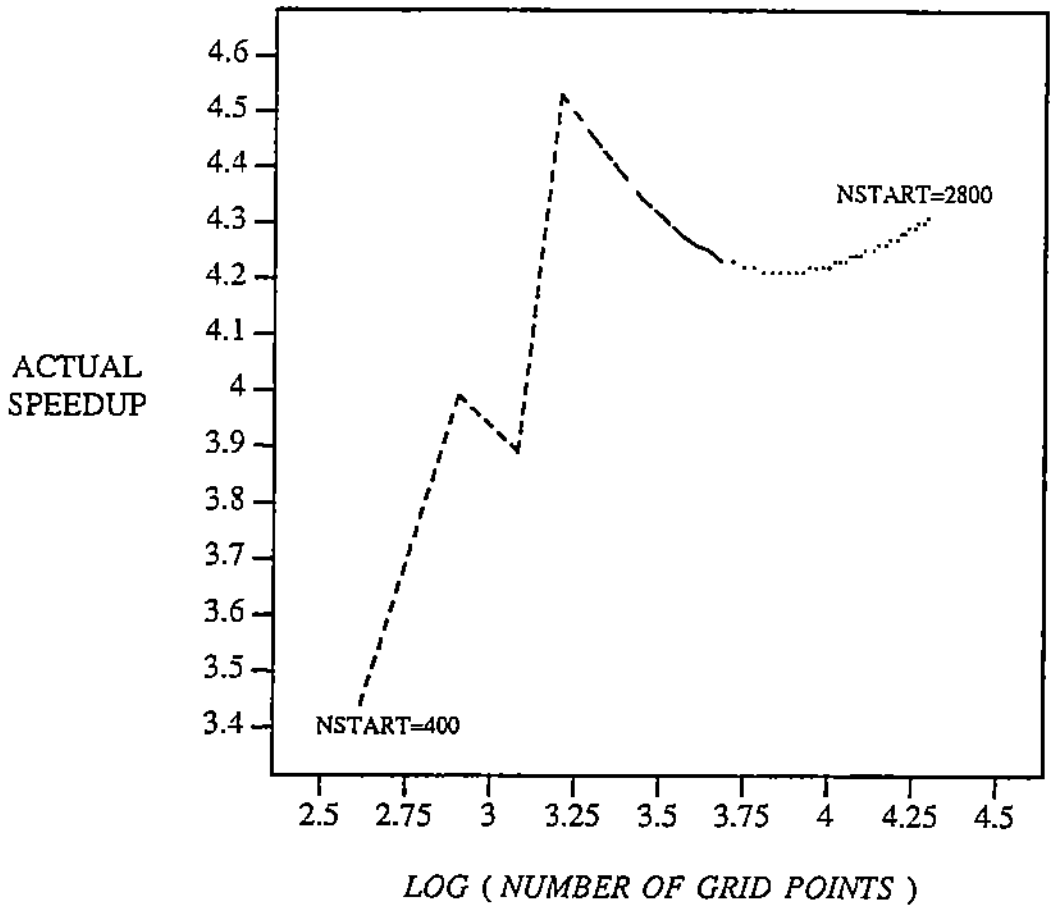

Figure 3c. The *speed up* function of the logarithm of the number of grid points for two values of NSTART, SOR convergence rate, and second order PDE discretization.

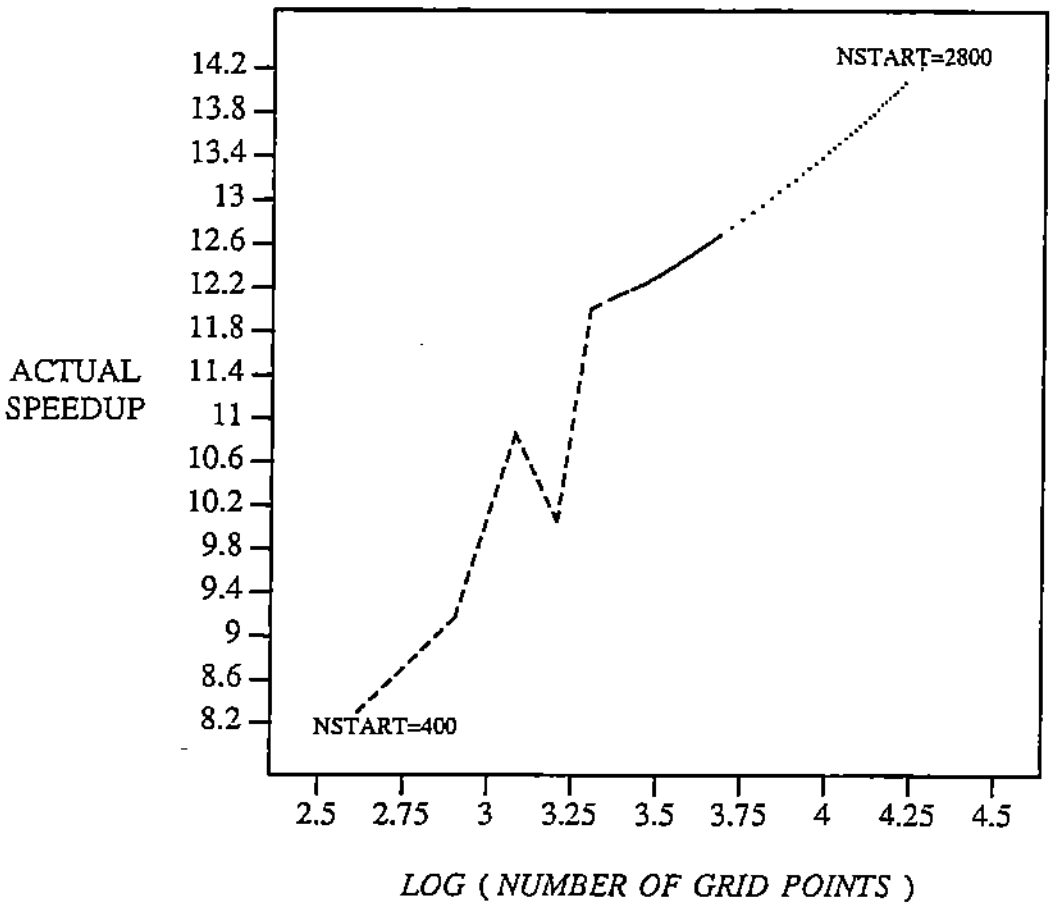

Figure 3d. The *speed up* function of the logarithm of the number of grid points for two values of NSTART, SOR convergence rate, and fourth order PDE discretization.

There is considerable general similarity among the plots of Figure 3 which leads us to conclude that the effectiveness of the two level iteration is not heavily dependent on the properties of the numerical methods underlying the computation.

Our earlier study of the effect of the machine communication characteristics, t_s (start up time) and t_c (time per unit message) had these two parameters varying nearly proportionally. We used the (t_s, t_c) pairs (400,4), (200,2) and (50,1). Figure 4 shows the actual speed ups obtained with the (t_s, t_c) pairs (50,4), (200,2) and (400,1) to further explore the effect of these parameters. Comparing Figure 4 with Figure 8 of [4] we observe that

- Increasing the time per unit message, t_c, significantly increases the speed up obtained from the two level algorithm.
- The start up time parameter, t_s, has a minor effect on the speed up obtained from the two level algorithm.

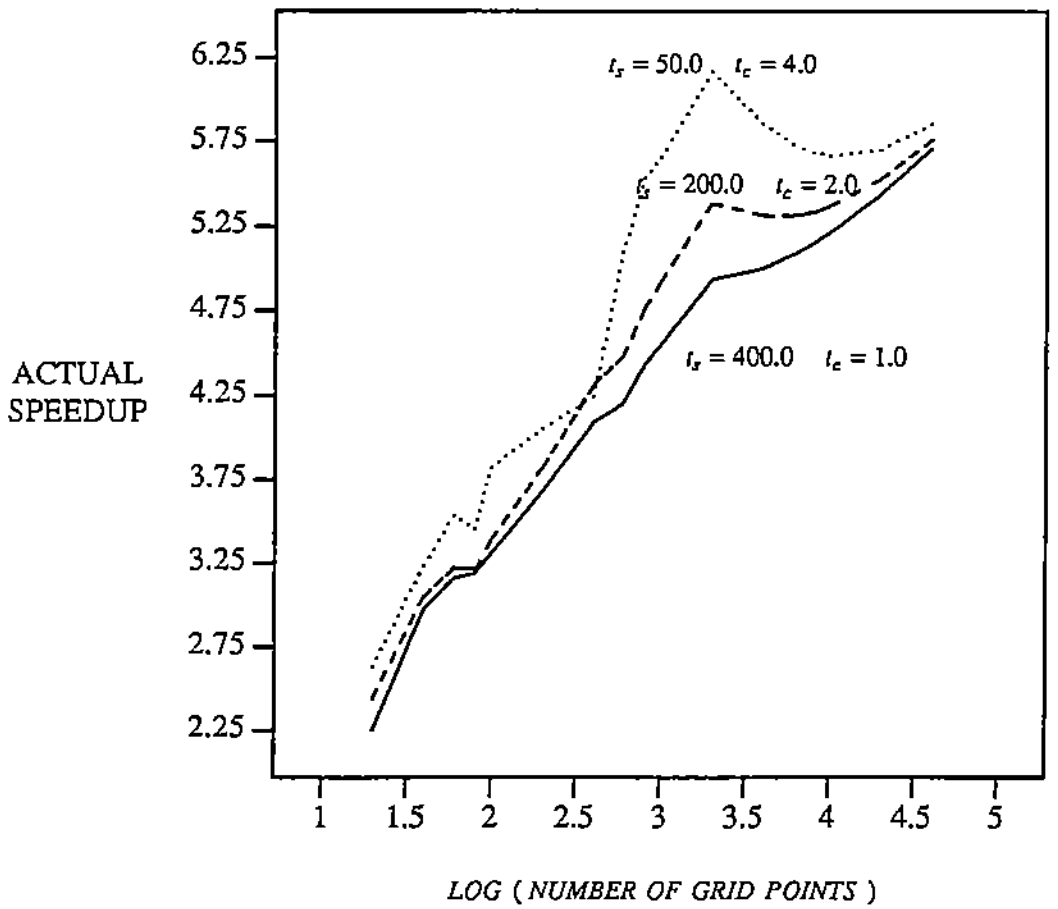

Figure 4 The *speed up* function of the logarithm of the number of grid points for three sets of machine parameters: t_s = communication start-up time, t_c = communication time per unit of message. The coefficient of variation of the execution time is $C_X = 0.2$.

7 REFERENCES

1. D.C. Marinescu and J.R. Rice, "On the effects of synchronization in parallel computation", CSD-TR-750, Computer Science Department, Purdue University, March 1988.

2. D.C. Marinescu and J.R. Rice, "Synchronization of nonhomogeneous parallel computations", *Parallel Processing for Scientific Computing*, (Rodrigue, ed.), SIAM, 1988 (to appear).

3. D.C. Marinescu and J.R. Rice, "Domain oriented analysis of PDE splitting algorithms", *Information Sciences*, 43, pp. 3–24, 1987.

4. D.C. Marinescu and J.R. Rice, "Analysis of a two level asynchronous algorithm for PDEs", in *Studies in Asynchronous Numerical Computations*, (M. Wright, ed.) North-Holland (1989).

5. J.R. Rice and D.C. Marinescu, "Analysis and modeling of Schwartz splitting algorithms for elliptic PDEs", in *Advances in Computer Methods for Partial Differential Equations*, VI (Steplemann and Vishnevetsky, eds.), IMACS, Rutgers University, pp. 1–6, 1987.

6. Y. Saad and M. Schultz, "Topological Properties of Hypercubes", *IEEE Trans. on Computers*, **37**, pp. 867-872, July 1988.

7. C.L. Seitz, "The cosmic cube", *Commun.*, ACM, **28**, pp. 22–31, 1985.

8. A.Y. Wu, "Embedding of tree networks into hypercubes", *J. Parallel Distributed Comput.*, **2**, pp. 238-249, 1985.